Traffic Flow Forecasting

Engineering Group Members:
Hammam Hraesha
Natnael Guesh Hagos
Markosyan Arman
Edgar Abasov

INTRODUCTION

- The PeMSD8 is a highway traffic dataset from California.
- Collected by the Caltrans Performance Measurement System (PeMS)
- Data collected in real time, every 30 seconds.
- The traffic data is aggregated into every 5-minute interval from the raw data.
- The system has more than 39,000 detectors deployed on the highway.
- Geographic information about the sensor stations is recorded in the datasets.
- The three relevant traffic measurements are: total flow, average speed, and average occupancy.

Data Characteristics

01 Temporal Data

- Shape of signal data: (17856, 170, 3)
- 17856 time steps → 5-minute intervals over ~62 days
- 170 sensors across the network
- 3 traffic features: flow, occupancy, speed
- Spans July–August 2016, consistent with the number of time steps

02 Spatial Data

- Each row represents a directed edge between two sensors
- Columns:
  - **from**, **to**: sensor node IDs
  - **Cost**: physical distance (e.g., in meters)

Data Preparation

Handling Missing Values

Apply linear interpolation per sensor & feature

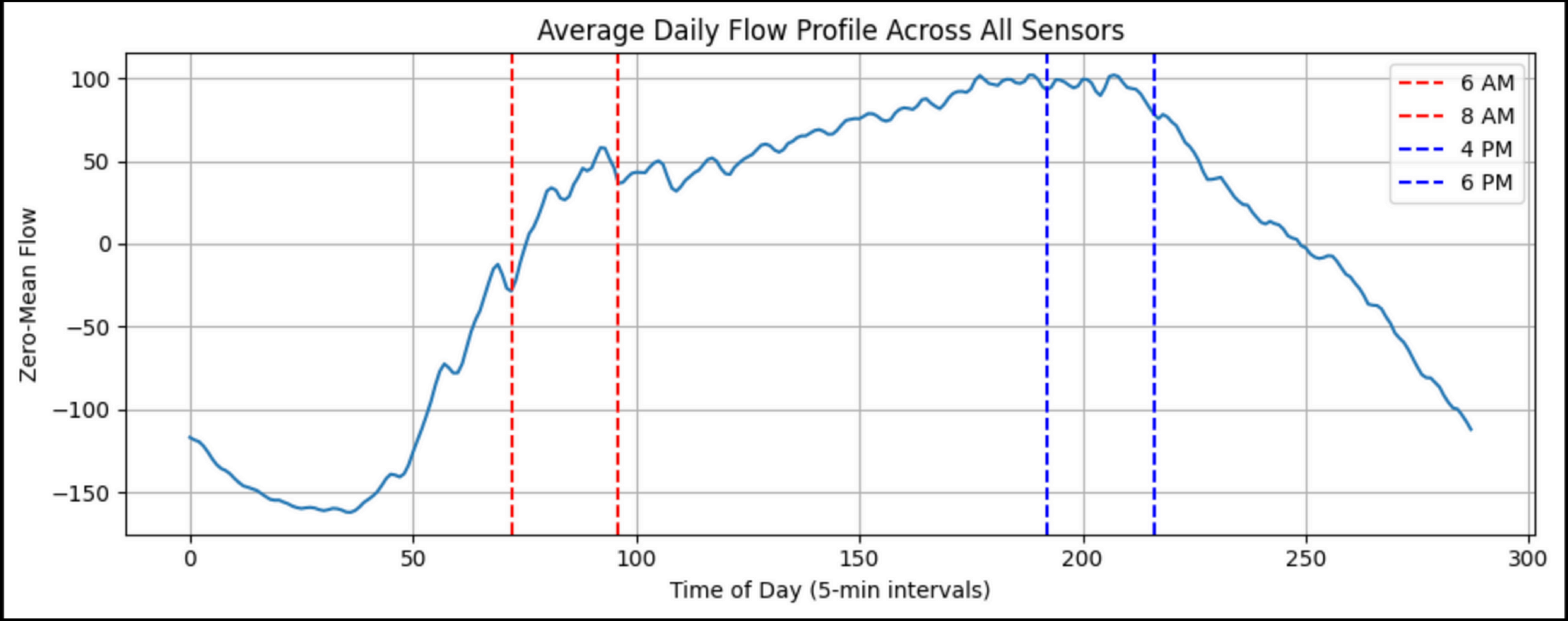
Handling Temporal Data

Zero-mean normalization

Handling Spatial Data

Build adjacency matrix

Visualizing Daily Traffic Flow



Traffic Flow Modeling

Models Used

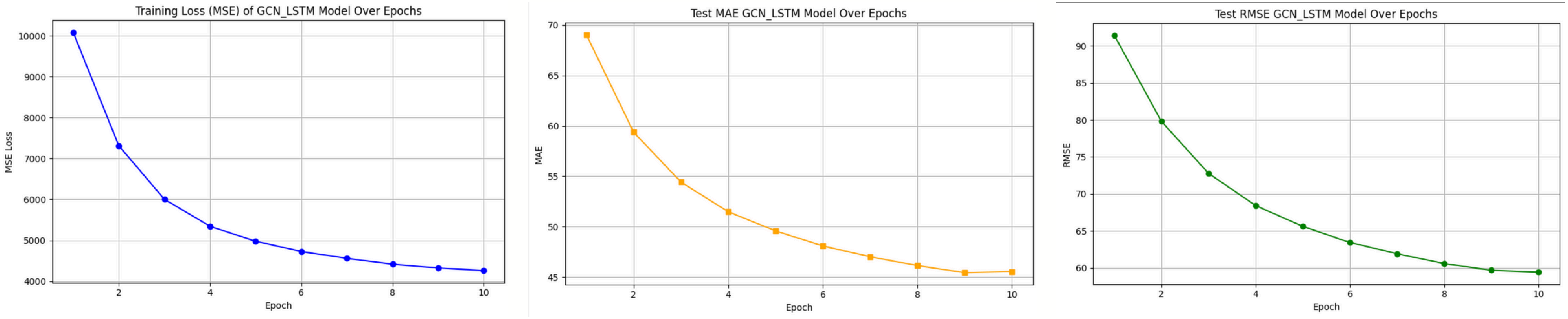
GCN_LSTM

Spatial relationships are modeled using three stacked Graph Convolutional Network (GCN) layers with hidden dimensions of 128 each. The spatially processed features are then passed through a two-layer LSTM with a hidden size of 128 and dropout of 0.2, applied independently to each sensor's temporal sequence.

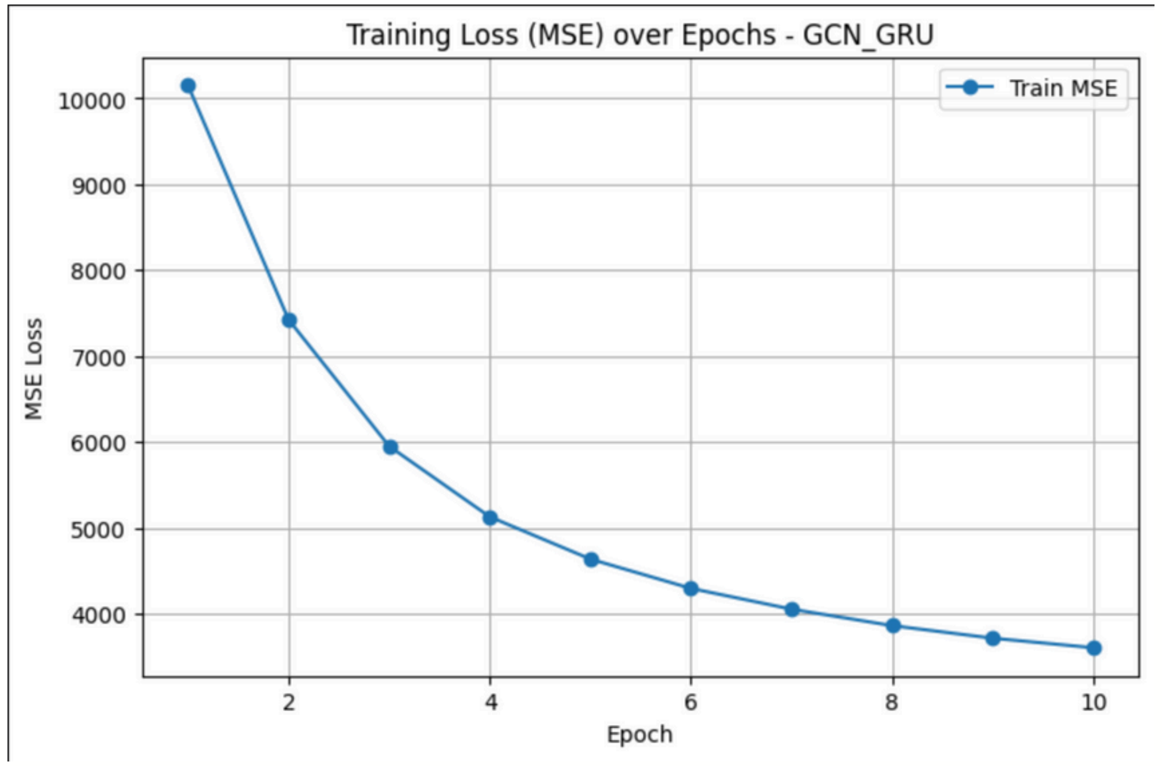
GCN_GRU

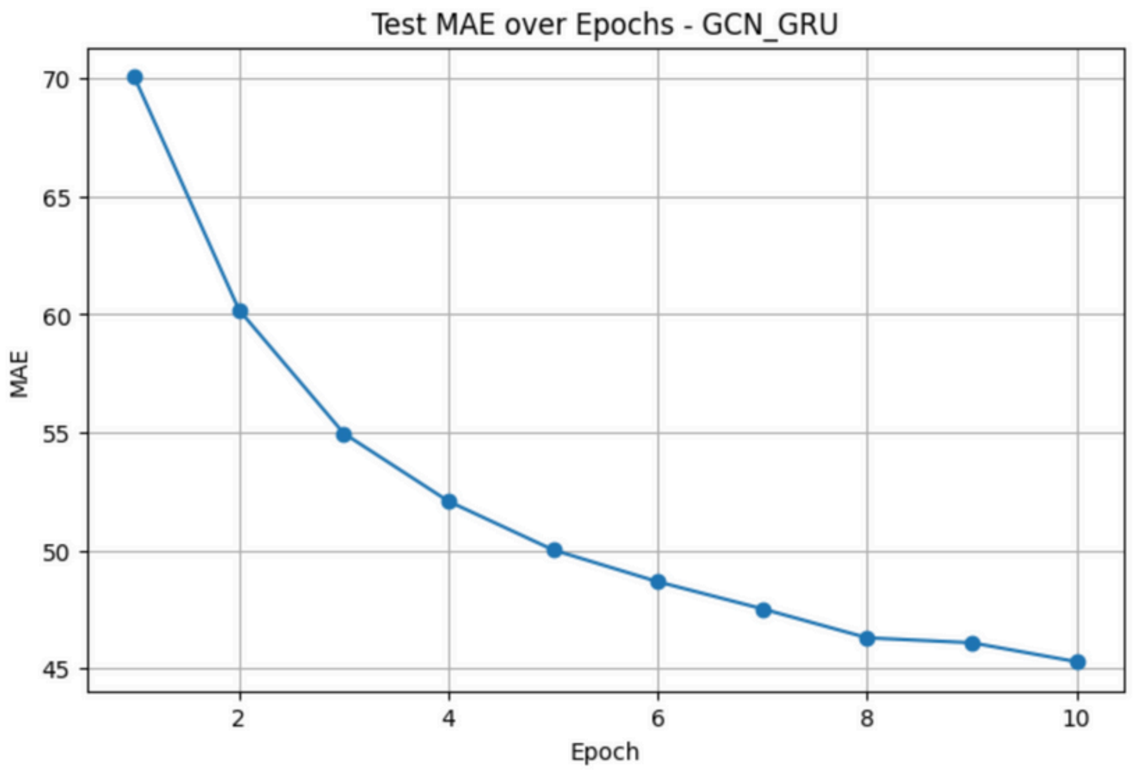
Follows a similar architecture to the previous model in terms of handling spatial relationships. The spatially processed data is then fed into a two-layer Gated Recurrent Unit (GRU) network with a hidden size of 128 and dropout of 0.2, applied independently to each sensor over time.

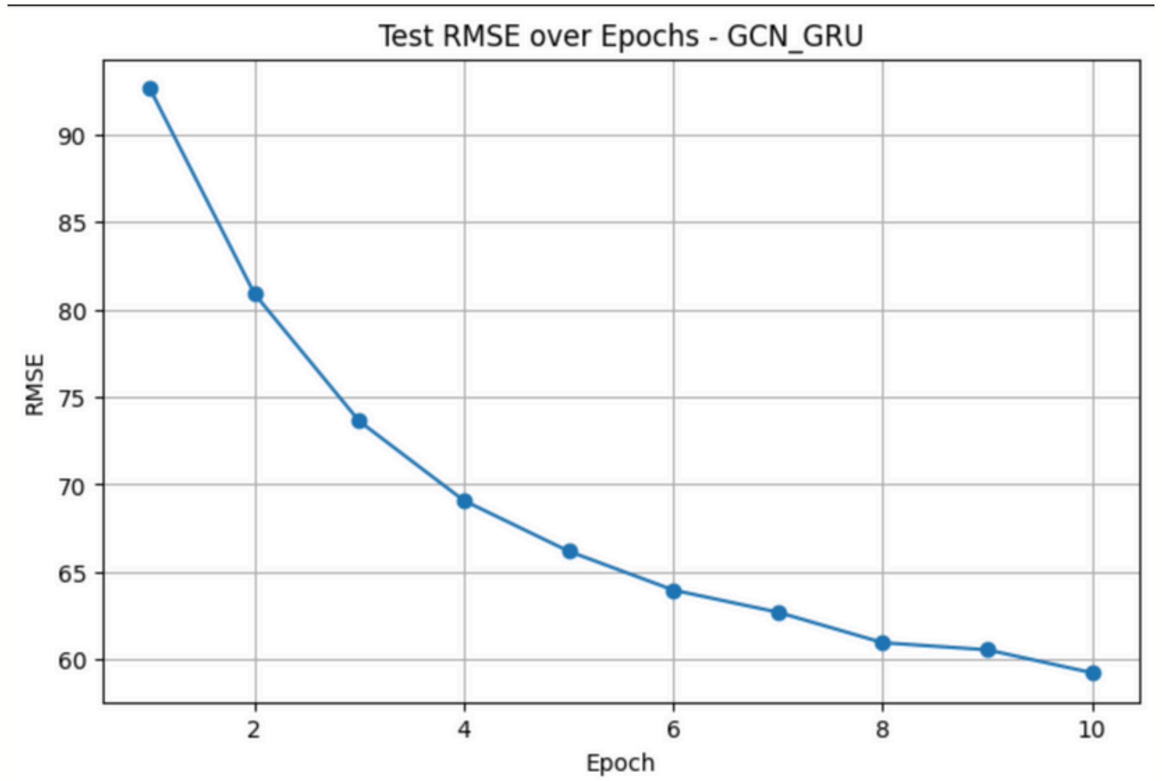
GCN_LSTM


Training Loss (MSE) of GCN_LSTM Model Over Epochs
MSE Loss
Epoch
10000
9000
8000
7000
6000
5000
4000
2
4
6
8
10
Test MAE GCN_LSTM Model Over Epochs
MAE
Epoch
70
65
60
55
50
45
Test RMSE GCN_LSTM Model Over Epochs
RMSE
Epoch
90
85
80
75
70
65
60

GCN_GRU


Training Loss (MSE) over Epochs - GCN_GRU
Train MSE
MSE Loss
Epoch
10000
9000
8000
7000
6000
5000
4000
2
4
6
8
10


Test MAE over Epochs - GCN_GRU
MAE
Epoch
70
65
60
55
50
45
2
4
6
8
10

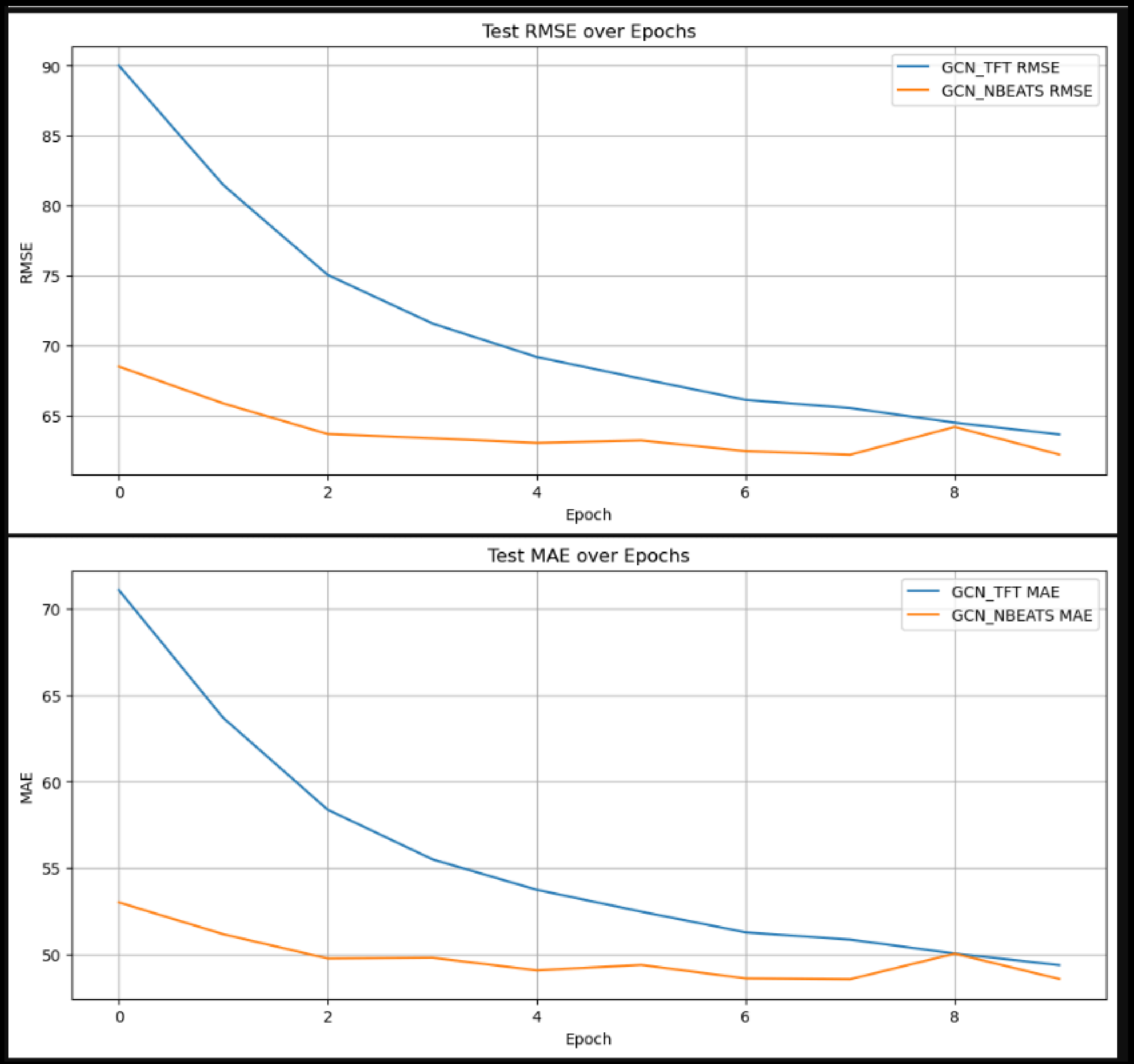

Test RMSE over Epochs - GCN_GRU
RMSE
Epoch
90
85
80
75
70
65
60
2
4
6
8
10

Graph-Based Traffic Forecasting

Models Compared

1. GCN-TFT
Combines Graph Convolution (GCN) with a simplified Transformer block for temporal modeling

2. GCN-NBEATS
Combines GCN with an MLP-style N-BEATS block (fully connected residual architecture)



XGBoost

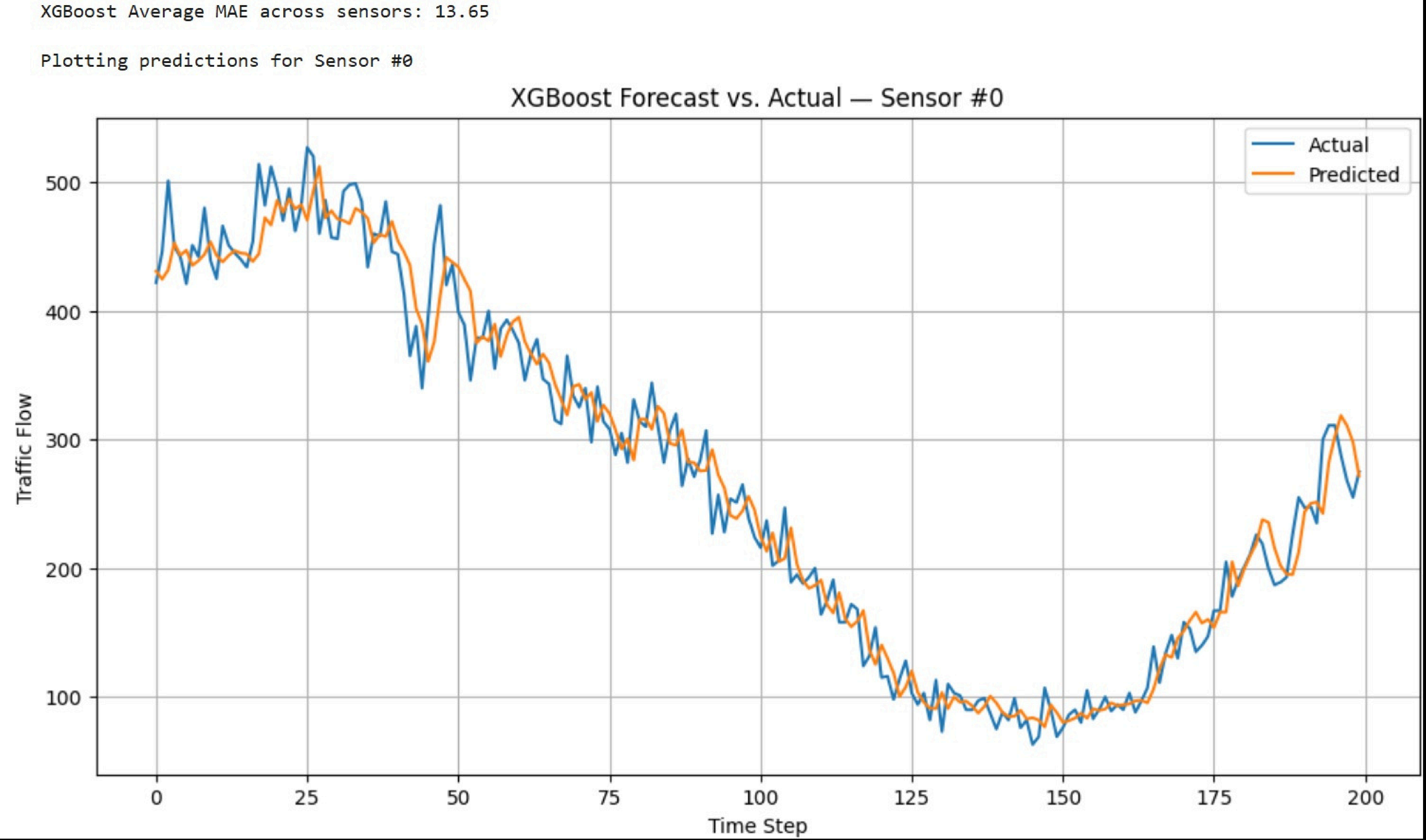
Fast and powerful

Decision-tree based regressor

Excellent for tabular time-series data

Performs well with minimal preprocessing

Weak at modeling long-term temporal dependencies



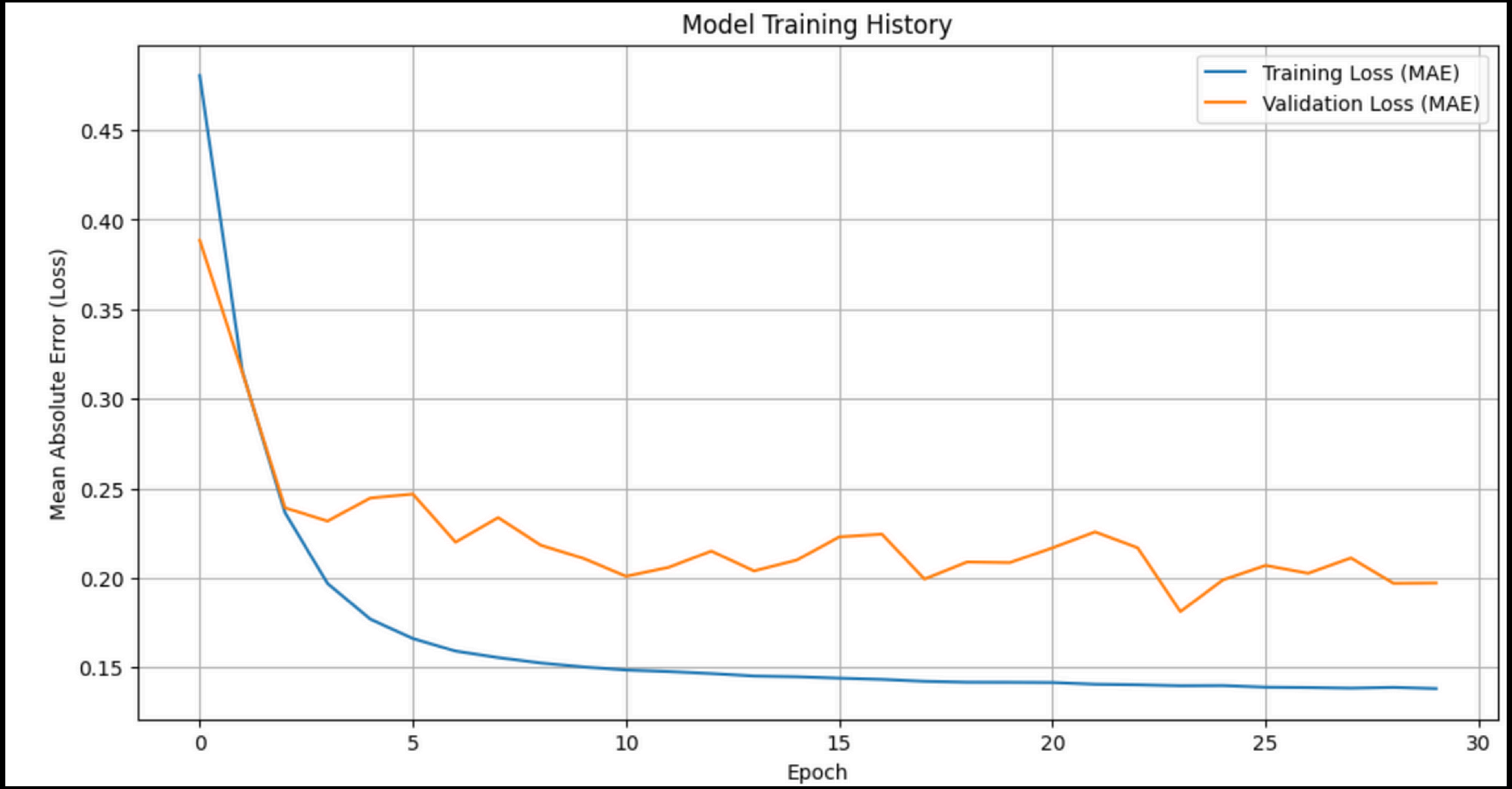
Lag-Llama

Llama based LLM adapted for time series forecasting

Captures complex patterns over long horizo

Supports fine-tuning forecasting

Requires GPU, more time and memory

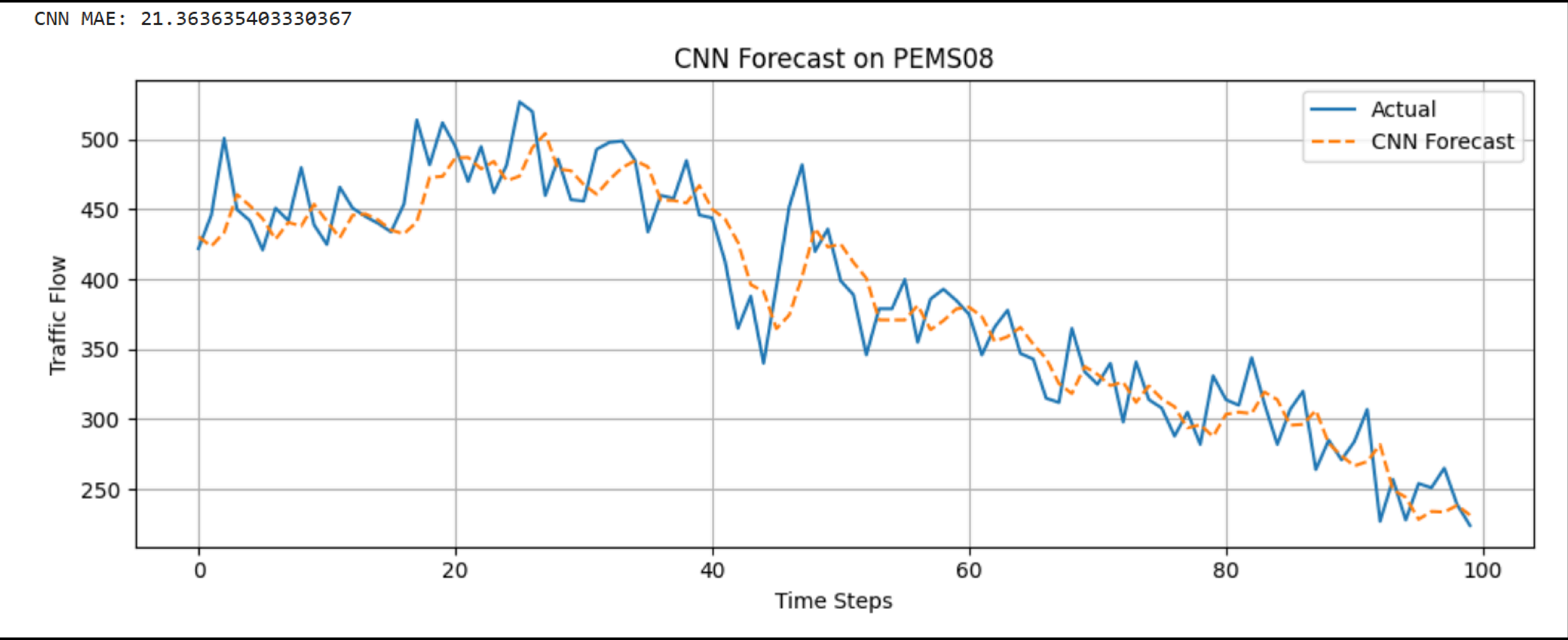


CNN

Efficient and lightweight

Struggles with long-range dependencies

Learns local temporal patterns
using sliding windows



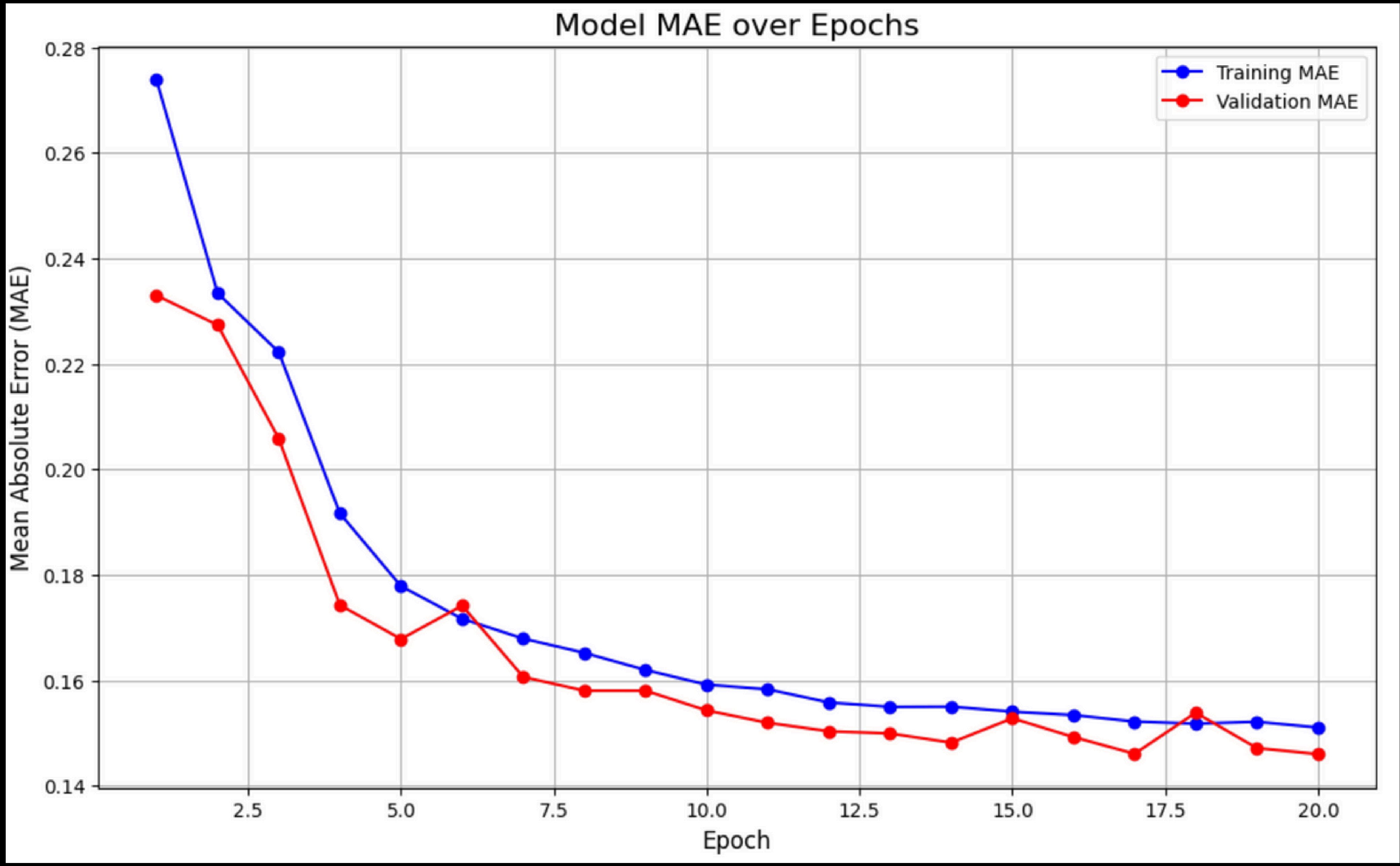
ASTGCN

Attention-based Spatio-Temporal Graph Convolutional Network

Combines graph convolutions, temporal CNNs and attention

Higher model complexity, but powerful

Models both spatial and temporal relationships



LGBMRegressor

- Data Approach
- Model Part
- Results Part

```
Mean Squared Error: 1568.7822127240997
Mean Absolute Error: 25.843579814013847
Root Mean Squared Error: 39.60785544212284
```

```python
def train_pipeline(X_train, Y_train, X_test, Y_test, save_models=True):
  print(f"X_train shape: {X_train.shape}")
  print(f"Y_train shape: {Y_train.shape}")
  print(f"X_test shape: {X_test.shape}")

  # Step 1: Feature Selection (reduce to 100 features)
  k_features = 100
  print(f"\nStarting feature selection to reduce to {k_features} features...")
  start_time_fs = time.time()

  # Flatten to 2D for feature selection: (14376, 12 × 170 × 3 = 6120)
  X_train_flat = X_train.reshape(X_train.shape[0], -1)  # (14376, 6120)
  X_test_flat = X_test.reshape(X_test.shape[0], -1)    # (3432, 6120)

  # Variance Thresholding
  vt_selector = VarianceThreshold(threshold=0.001)
  X_train_pre_selected = vt_selector.fit_transform(X_train_flat)
  X_test_pre_selected = vt_selector.transform(X_test_flat)
  vt_selected_indices = vt_selector.get_support(indices=True)
  print(f"VarianceThreshold removed {X_train_flat.shape[1] - X_train_pre_selecte
  print(f"Shape after VarianceThreshold: {X_train_pre_selected.shape}")

  # F-regression: Use last timestep’s flow (Y_train[:, -1, :, 0]) for scoring
  print("####### Y_train", {Y_train.shape})
  Y_train_flow = Y_train[:, -1, :, 0]  # (14376, 170)
  Y_train = Y_train.reshape((Y_train.shape[0], 12, 170, 3))
  print("####### Y_train", {Y_train.shape})
  print("####### Y_train_flow", {Y_train_flow.shape})
  feature_scores_sum = np.zeros(X_train_pre_selected.shape[1])

  for i in tqdm(range(Y_train_flow.shape[1]), desc="Aggregating scores per targe
      f_scores, _ = f_regression(X_train_pre_selected, Y_train_flow[:, i])
      feature_scores_sum += f_scores
  top_k_relative_indices = np.argsort(feature_scores_sum)[::-1][:k_features]
  final_selected_indices = vt_selected_indices[top_k_relative_indices]
```

```python
def predict_pipeline(X_new, model_dir="saved_models"):
    """
    Prediction pipeline using saved models

    Args:
        X_new: New data for prediction (samples, 6124)
        model_dir: Directory with saved models

    Returns:
        predictions: Predictions for new data
    """

    print("=== Prediction Pipeline ===")
    print(f"X_new shape: {X_new.shape}")

    # Load models and feature indices
    models, feature_indices = load_everything(model_dir)

    # Apply feature selection
    X_new_reduced = X_new[:, feature_indices]
    print(f"X_new reduced shape: {X_new_reduced.shape}")

    # Make predictions
    predictions = predict_all(models, X_new_reduced)

    print(f"Predictions shape: {predictions.shape}")
    return predictions
```

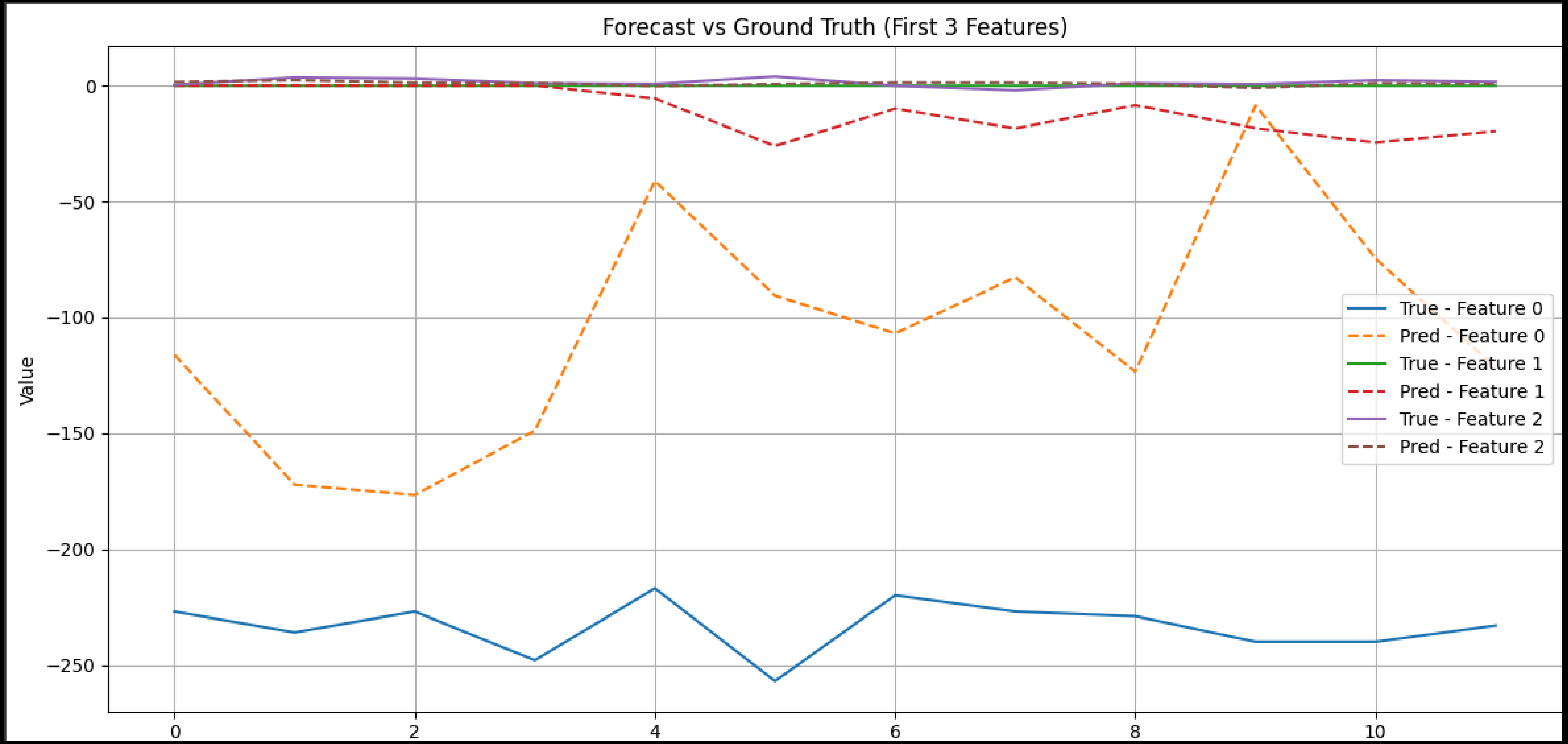
Chronos

Pretrained Models for Probabilistic Time Series Forecasting

```python
from chronos import ChronosPipeline

pipeline = ChronosPipeline.from_pretrained(
    "amazon/chronos-t5-small",
    device_map="cpu",
    torch_dtype=torch.float32,
)
```

```python
mae = mean_absolute_error(y_true_cut.numpy(), y_pred

mse = mean_squared_error(y_true_cut.numpy(), y_pred)
rmse = np.sqrt(mse)

print(f"MAE: {mae:.4f}")
# print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")
```

```
MAE: 58.3238
RMSE: 92.6407
```



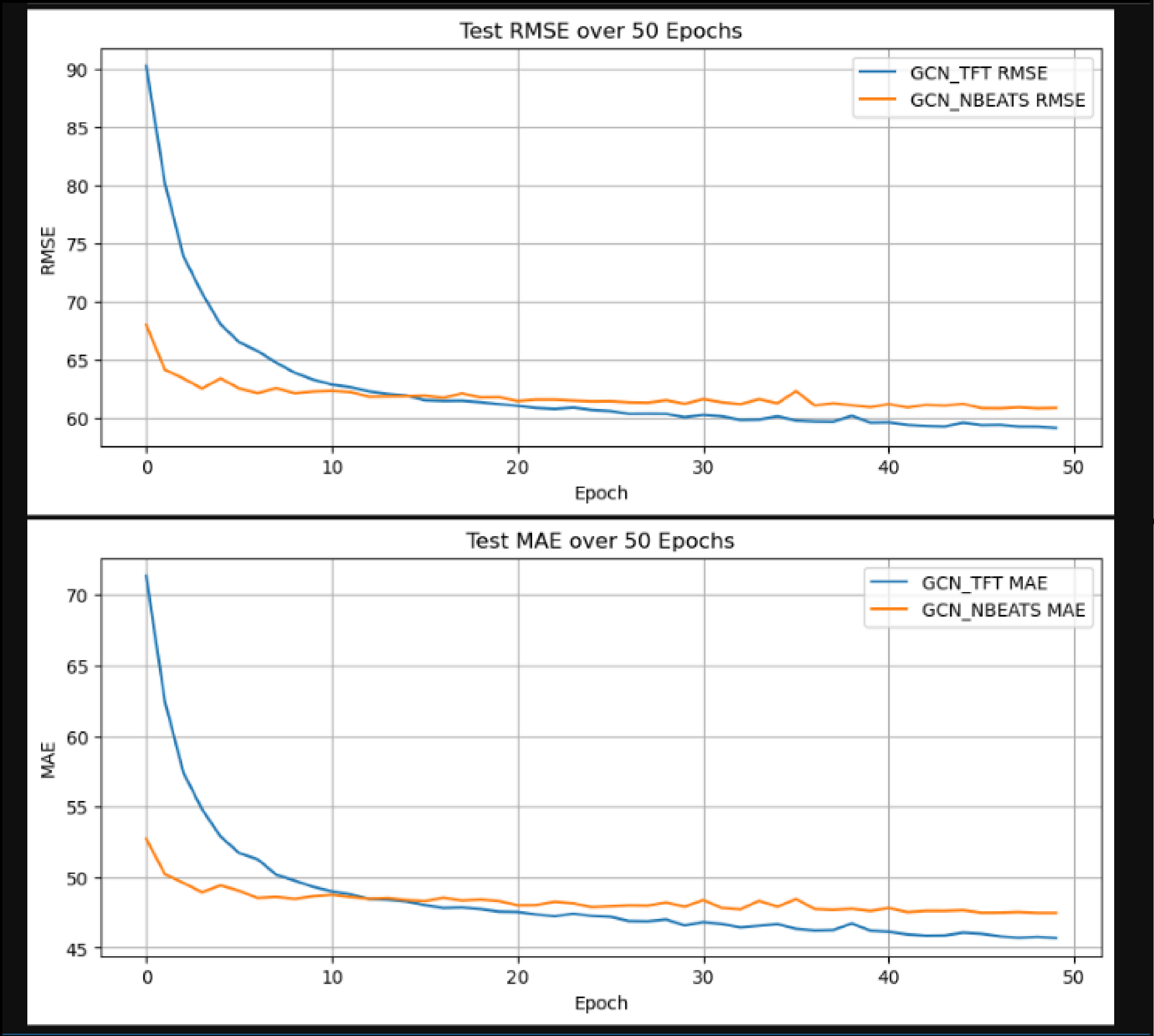
Models Evaluation

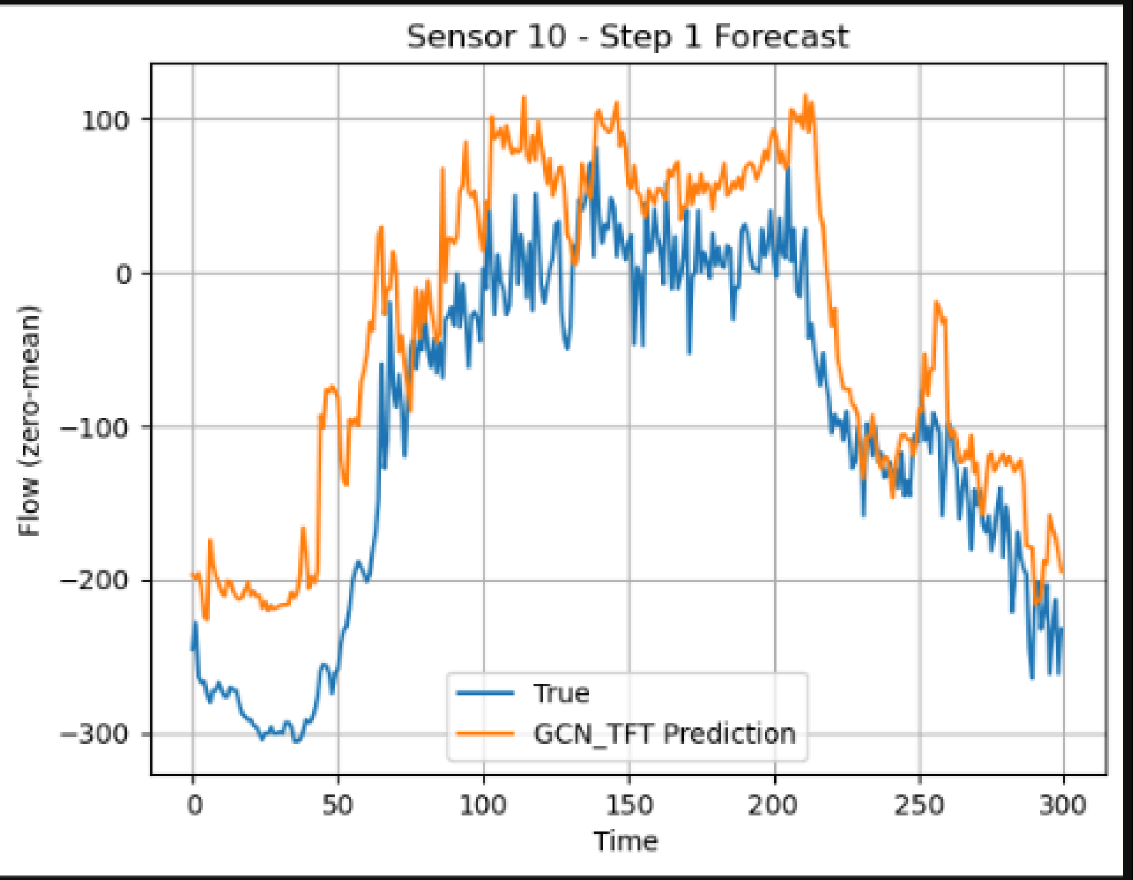
Models	GCN_LSTM	GCN_GRU	GCN-TFT	LGBM	GCN-NBEATS	XGBoost	Llama	ASTGCN	CNN	Chronos
MAE (%)	45.5	45.2	45.3	25.84	46.2	13.65	13.8	14.6	21.33	58.32
RMSE (%)	59.4	59.2	59	39.60	60	18.9	19.72	15.15	31.1	92.64
Epochs/ boosting rounds	10	10	10	100 rounds	10	100 rounds	30	20	10	10

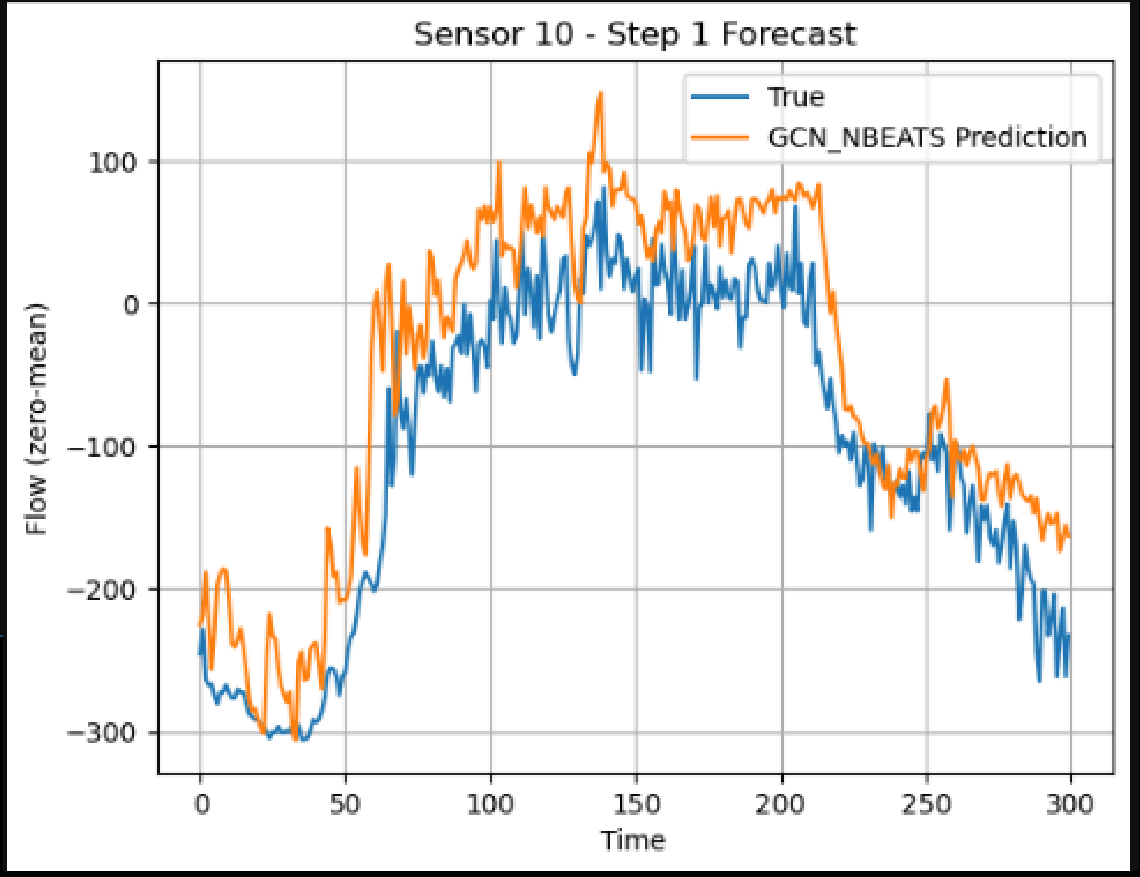
Challenges

GCN-TFT vs GCN-NBEATS (50 Epochs)

- Training time
- Data Preparation
- Resource limitations
- Expertise in the literature







Thank You

https://github.com/Metaphysicist1/Traffic-Data-Time-Series-Forecasting